Online Payment Fraud Detection using Machine Learning in Python

Last Updated : 21 Nov, 2022

As we are approaching modernity, the trend of paying online is increasing tremendously. It is very beneficial for the buyer to pay online as it saves time, and solves the problem of free money. Also, we do not need to carry cash with us. But we all know that **Good thing are accompanied by bad things**.
The online payment method leads to fraud that can happen using any payment app. That is why Online Payment Fraud Detection is very important.

Online Payment Fraud Detection using Machine Learning in Python

Here we will try to solve this issue with the help of machine learning in Python.
The dataset we will be using have these columns –

Feature	Description
step	tells about the unit of time
type	type of transaction done
amount	the total amount of transaction
nameOrg	account that starts the transaction
oldbalanceOrg	Balance of the account of sender before transaction
newbalanceOrg	Balance of the account of sender after transaction
nameDest	account that receives the transaction
oldbalanceDest	Balance of the account of receiver before transaction
newbalanceDest	Balance of the account of receiver after transaction
isFraud	The value to be predicted i.e. 0 or 1

Importing Libraries and Datasets

The libraries used are :

- Pandas: This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
- Seaborn/Matplotlib: For data visualization.
- Numpy: Numpy arrays are very fast and can perform large computations in a very short time.

- Python3

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

The dataset includes the features like type of payment, Old balance , amount paid, name of the destination, etc.

- Python3

```
data = pd.read_csv('new_data.csv')
data.head()
```

Output :

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	358	PAYMENT	2739.93	C1139154882	117230.00	114490.07	M2138213924	0.00	0.00	0
1	371	CASH_OUT	124119.87	C603024364	303076.00	178956.13	C1110031759	0.00	124119.87	0
2	129	CASH_IN	47715.14	C1297774858	21073457.15	21121172.29	C1790183137	838365.79	790650.65	0
3	329	CASH_IN	65739.60	C1830063487	16738153.92	16803893.52	C790107917	371889.30	306149.70	0
4	187	CASH_OUT	270253.77	C1879008092	0.00	0.00	C131873960	2167102.08	2437355.85	0

To print the information of the data we can use data.info() command.

- Python3

```
data.info()
```

Output :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16000 entries, 0 to 15999
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   step            16000 non-null  int64
 1   type            16000 non-null  object
 2   amount          16000 non-null  float64
 3   nameOrig        16000 non-null  object
 4   oldbalanceOrg   16000 non-null  float64
 5   newbalanceOrig  16000 non-null  float64
 6   nameDest        16000 non-null  object
 7   oldbalanceDest  16000 non-null  float64
 8   newbalanceDest  16000 non-null  float64
 9   isFraud         16000 non-null  int64
dtypes: float64(5), int64(2), object(3)
```

Let's see the mean, count , minimum and maximum values of the data.

- Python3

```
data.describe()
```

Output :

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
count	16000.000000	1.600000e+04	1.600000e+04	1.600000e+04	1.600000e+04	1.600000e+04	16000.000000
mean	306.068562	8.196301e+05	1.223819e+06	5.103682e+05	8.285281e+05	1.258598e+06	0.500000
std	194.036242	1.901944e+06	3.279212e+06	2.539758e+06	3.447489e+06	4.009254e+06	0.500016
min	1.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
25%	161.000000	3.575912e+04	1.057991e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
50%	282.000000	1.717888e+05	1.169403e+05	0.000000e+00	0.000000e+00	1.137627e+05	0.500000
75%	411.000000	5.362124e+05	7.643284e+05	0.000000e+00	4.922000e+05	1.077581e+06	1.000000
max	743.000000	5.778780e+07	5.958504e+07	4.958504e+07	2.362305e+08	2.367265e+08	1.000000

Data Visualization

In this section, we will try to understand and compare all columns.

Let's count the columns with different datatypes like Category, Integer, Float.

- Python3

```
obj = (data.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:", len(object_cols))
```

```
int_ = (data.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:", len(num_cols))

fl = (data.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:", len(fl_cols))
```

Output :

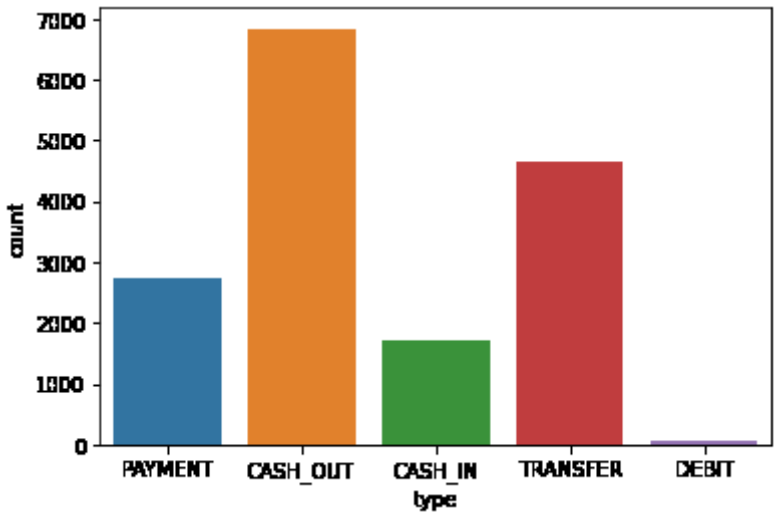
```
Categorical variables: 3

Integer variables: 2

Float variables: 5
```

Let's see the count plot of the Payment type column using Seaborn library.

- Python3

```
sns.countplot(x='type', data=data)
```

Output :

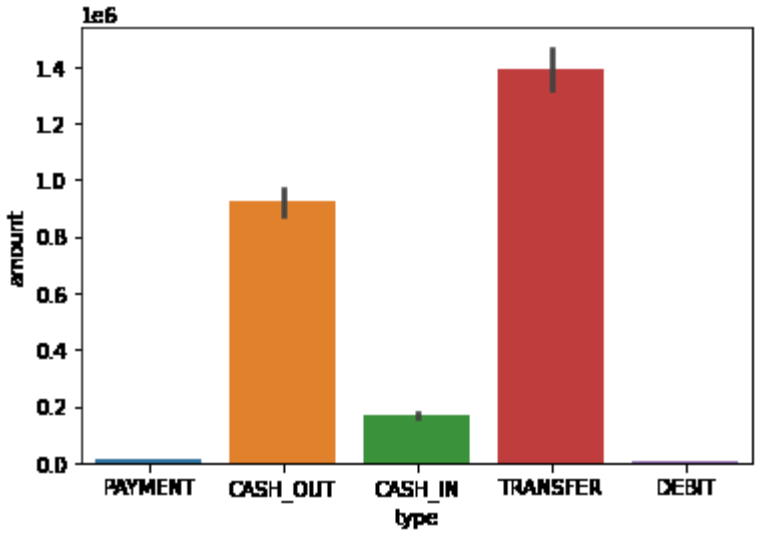


We can also use the bar plot for analyzing **Type** and **amount** column simultaneously.

- Python3

```
sns.barplot(x='type', y='amount', data=data)
```

Output :



Both the graph clearly shows that mostly the type cash_out and transfer are maximum in count and as well as in amount.

Let's check the distribution of data among both the prediction values.

- Python3

```
data['isFraud'].value_counts()
```

Output :

```
0    8000
1    8000
```

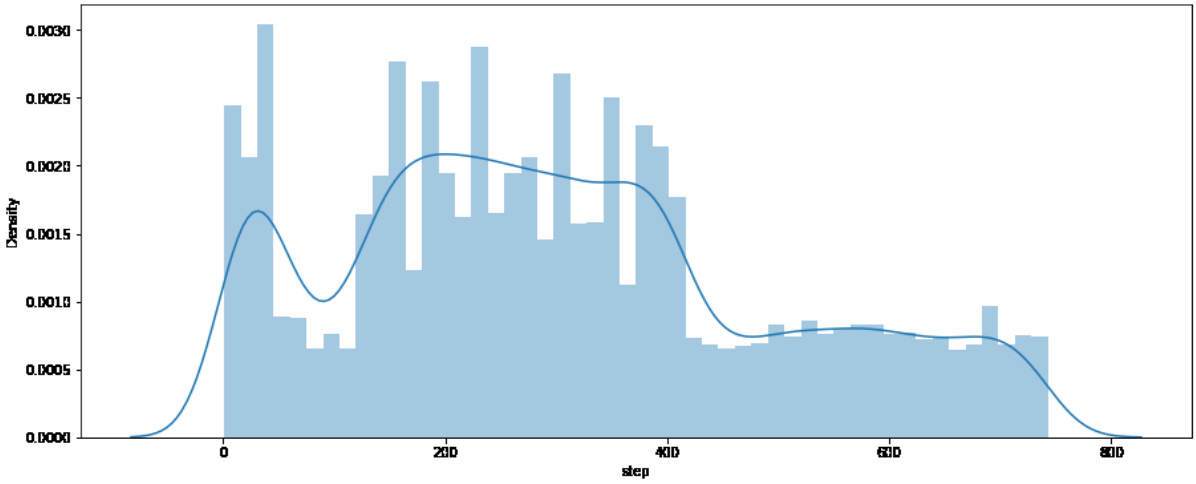
The dataset is already in same count. So there is no need of sampling.

Now let's see the distribution of the step column using distplot.

- Python3

```
plt.figure(figsize=(15, 6))
sns.distplot(data['step'], bins=50)
```

Output :



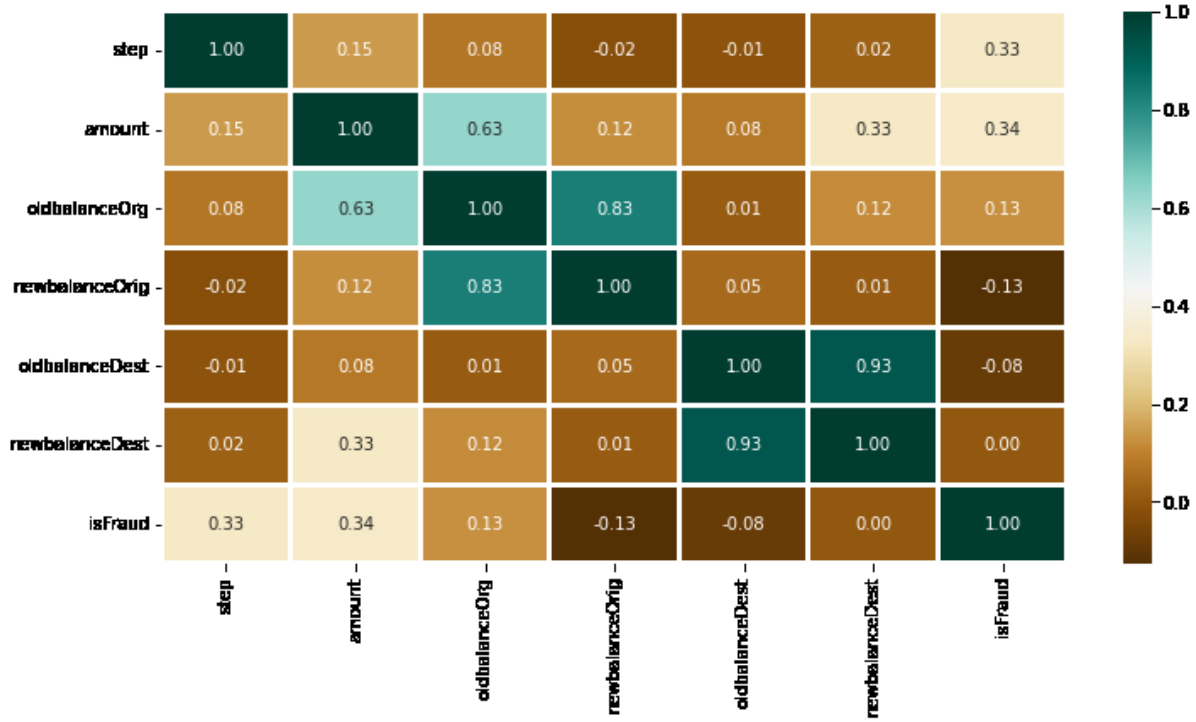
The graph shows the maximum distribution among 200 to 400 of step.

Now, Let's find the correlation among different features using Heatmap.

- Python3

```
plt.figure(figsize=(12, 6))
sns.heatmap(data.corr(),
            cmap='BrBG',
            fmt='.2f',
            linewidths=2,
            annot=True)
```

Output :



Data Preprocessing

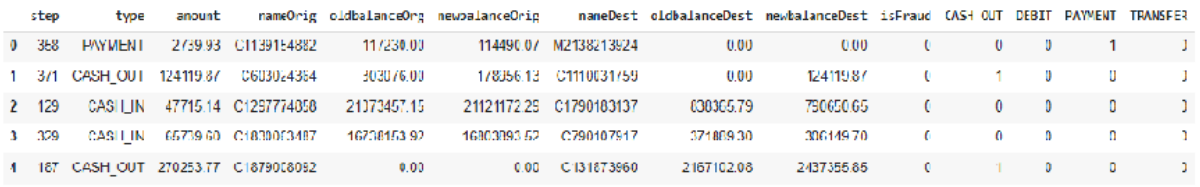
This step includes the following :

- Encoding of Type column
- Dropping irrelevant columns like nameOrig, nameDest
- Data Splitting

- Python3

```
type_new = pd.get_dummies(data['type'], drop_first=True)
data_new = pd.concat([data, type_new], axis=1)
data_new.head()
```

Output:



	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	CASH_OUT	DEBIT	PAYMENT	TRANSFER
0	368	PAYMENT	2739.93	C1139154882	11230.00	114490.07	M2138213924	0.00	0.00	0	0	0	1	0
1	371	CASH_OUT	124119.87	C6030c4364	903076.00	178956.13	C1110031750	0.00	124119.87	0	1	0	0	0
2	129	CASH_IN	47715.14	C1257774058	21073457.15	21121172.29	C1790183137	630305.79	750650.65	0	0	0	0	0
3	329	CASH_IN	66779.60	C1870003487	16736153.92	16803093.52	C790107917	371809.30	306149.70	0	0	0	0	0
4	167	CASH_OUT	270253.77	C1879008082	0.00	0.00	C1316873960	2167102.08	2437355.85	0	1	0	0	0

Once we done with the encoding, now we can drop the irrelevant columns. For that, follow the code given below.

- Python3

```
X = data_new.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
y = data_new['isFraud']
```

Let's check the shape of extracted data.

- Python3

```
X.shape, y.shape
```

Output:

```
((16000, 10), (16000,))
```

Now let's split the data into 2 parts : Training and Testing.

- Python3

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
```

Model Training

As the prediction is a classification problem so the models we will be using are :

- LogisticRegression : It predicts that the probability of a given data belongs to the particular category or not.
- XGBClassifier : It refers to Gradient Boosted decision trees. In this algorithm, decision trees are created in sequential form and weights are assigned to all the independent variables which are then fed into the decision tree which predicts results.
- SVC : SVC is used to find a hyperplane in an N-dimensional space that distinctly classifies the data points. Then it gives the output according the most nearby element.
- RandomForestClassifier : Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. Then, it collects the votes from different decision trees to decide the final prediction.

Let's import the modules of the relevant models.

- Python3

```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score as ras
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

Once done with the importing, Let's train the model.

- Python3

```
models = [LogisticRegression(), XGBClassifier(),
          SVC(kernel='rbf', probability=True),
          RandomForestClassifier(n_estimators=7,
                                 criterion='entropy',
                                 random_state=7)]

for i in range(len(models)):
    models[i].fit(X_train, y_train)
    print(f'{models[i]} : ')

    train_preds = models[i].predict_proba(X_train)[:, 1]
    print('Training Accuracy : ', ras(y_train, train_preds))
```

```
    y_preds = models[i].predict_proba(X_test)[:, 1]
    print('Validation Accuracy : ', ras(y_test, y_preds))
    print()
```

Output:

```
LogisticRegression() :
Training Accuracy :  0.9610946236487818
Validation Accuracy :  0.9650647516187905

XGBClassifier() :
Training Accuracy :  0.9990647916240432
Validation Accuracy :  0.9988292242028274

SVC(probability=True) :
Training Accuracy :  0.9577130392435476
Validation Accuracy :  0.9610511096110737

RandomForestClassifier(criterion='entropy', n_estimators=7, random_state=7) :
Training Accuracy :  0.9999942442337746
Validation Accuracy :  0.9966858546463663
```

Model Evaluation

The best-performed model is **XGBClassifier.** Let's plot the Confusion Matrix for the same.

- Python3

```
from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(models[1], X_test, y_test)
plt.show()
```

Output: